IKANALM

Modern Mainframe development and Application Lifecycle Management

Cost-effective and easy to implement Enterprise-wide ALM for both mainframe and non-mainframe environments

Table of contents

Management summary

As a CIO, you are confronted with major challenges: you have to ensure business continuity and protect your company's former IT investments, and at the same time, you need to innovate, master the risks and be cost-efficient. Unfortunately, the budgets to accomplish all that keep on getting smaller.

Software applications play a key role in your company's revenue plans, but they are becoming more complex and have to keep pace with the ever changing business needs.

Many of you have a huge mainframe legacy, supporting the key operations of your company. At the same time, innovation drives you to other platforms, especially for customer and end-user applications. Today, it is unimaginable that applications are not available through the Web or via mobile devices.

Protecting that mainframe legacy to ensuring business continuity and at the same time being innovative by supporting Web and mobile applications, at an acceptable cost and while mastering the risks, is no child's play.

Mainframes are mainly used as back-end machines, and the front-end is mostly served through the Web or mobile devices.

That means that you not only have to manage your mainframe legacy, but also the new frontends and, most of all, the dependencies between those two.

Applications play a major role in Revenue Growth

Cost Efficiency

Smaller Budgets

Shorter and more complex Application Life Cycle

An additional serious concern is where to find the resources that still have mainframe skills or who are willing to acquire them?

The answer lies in understanding the changing role and use of mainframes, and in finding out how today's technology can be used to support both mainframes and distributed environments.

This is where Application Lifecycle Management (ALM) enters the game. ALM is used to manage the different steps in your application's lifecycle. It is easy to find mainframe-based or PC-based ALM solutions. However, finding one and the same ALM solution that handles both environments is almost impossible.

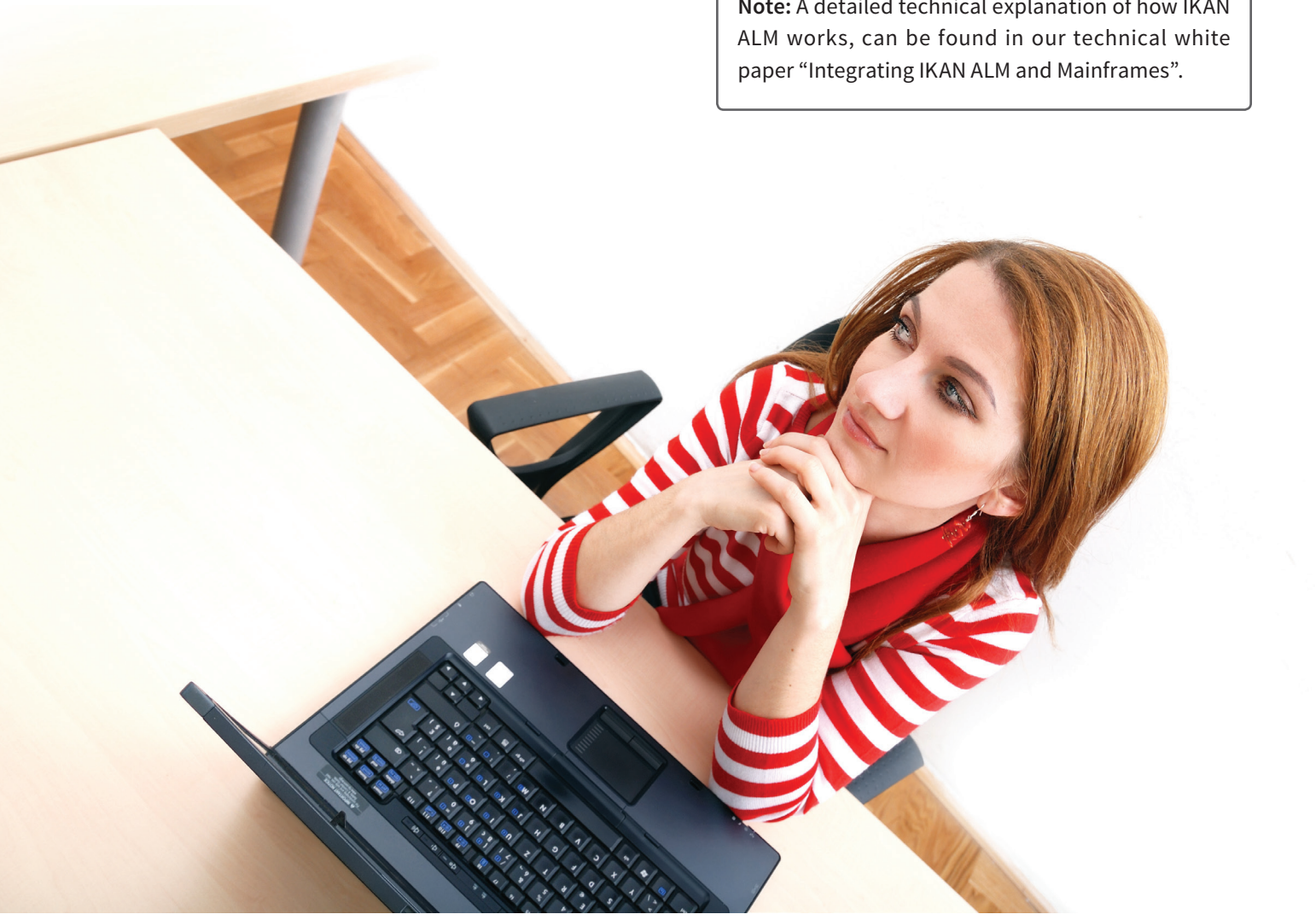
Almost … as there exists IKAN ALM.

IKAN ALM is IKAN's flagship product that takes care of all previously mentioned challenges. It is innovative, protects your legacy, masters the risks and is cost-efficient. On top of that, it supports both mainframe and non-mainframe systems.

The objective of this white paper is to explain in more detail how IKAN ALM can radically improve your mainframe lifecycle management, and especially how the developed applications can be deployed on the mainframe.

First, we will explain how traditional development is done on a mainframe and how the same work can be done using the currently available PC-based tooling. Secondly, we will explain what enterprise-wide Application Lifecycle Management stands for, and, finally, we will show how IKAN ALM can be used as a single point of control to protect your mainframe investments and combine the best of both worlds.

> **Note:** A detailed technical explanation of how IKAN ALM works, can be found in our technical white paper "Integrating IKAN ALM and Mainframes".



Traditional or Modern Mainframe Development

Conceptually, there is no difference in how a developer develops a program for mainframes, Windows, Linux or mobile devices:

1. **The software program needs to be developed, needs to be built/compiled and needs to run.**
2. **To develop a program, the developer needs a programming language and an editor.**
3. **Once it is written, the program needs to be saved and/or versioned, and compiled.**

The following lists up the main differences between developing programs for mainframe on a mainframe or on PC:

	Mainframe	PC
Programming languages	On mainframes, COBOL and PL/1 are the most popular programming languages	For Windows, Linux and Mobile, .NET and Java are widely used.
Program editor	IBM z/OS developers use ISPF as editor or development environment	.NET users use Visual Studio and Java developers mostly use an Eclipse-based editor
File system	IBM z/OS programmers save their programs in a PDS (Partitioned Data Set)	Under Windows a standard Windows directory is used
Versioning system	On IBM z/OS, CA-Panvalet, CA-Librarian, Serena ChangeMan, MSP Data Manager and IBM SCLM are used as version control systems	.NET developers use Visual Source Safe or Team Foundation Server, and Java developers use CVS, SUBVERSION or GIT
Compile procedure	On IBM z/OS, programs are compiled to translate source programs into load modules	For .NET and Java, the code is built to obtain an executable

In the next sections, we will explain how a developer works on the mainframe and how that same work can be done using PC-based tooling. Each of those sections covers a specific part of the developer's role.

Once the development is done, the developer's job is finished and the ALM system will take over to do the final compile on the mainframe. IKAN ALM can help you to complete the application lifecycle, by providing fully automated services to compile/build and deploy/promote to test and production environments.

Program Editor

On the mainframe

On IBM z/OS, the most commonly used editor is ISPF (Interactive System Productivity Facility). ISPF provides developers features for application development and for administering the z/OS operating system. The features include:

- Browse - for viewing data sets and Partitioned Data Set (PDS) members
- Edit - for editing data sets and PDS members
- Utilities - for performing data manipulation operations, such as:
  - Data Set List - which allows the User to list and manipulate (copy, move, rename, print, catalog, delete, etc.) files (called "data sets" in the z/OS environment).
  - Member List - for similar manipulations of members of PDSs.
  - Search facilities for finding modules or text within members or data sets.
  - Compare facilities for comparing members or data sets.

ISPF is mostly used as development environment or editor for writing COBOL, PL/1 programs and mainframe scripts called JCL (Job Control Language).

Non-mainframe alternative

A today's alternative for writing COBOL, PL/1 or even JCL ISPF, are Eclipse-based editors running on a PC.

Basically, an Eclipse-based or .NET editor offers the same functionality as ISPF and some additional benefits like:

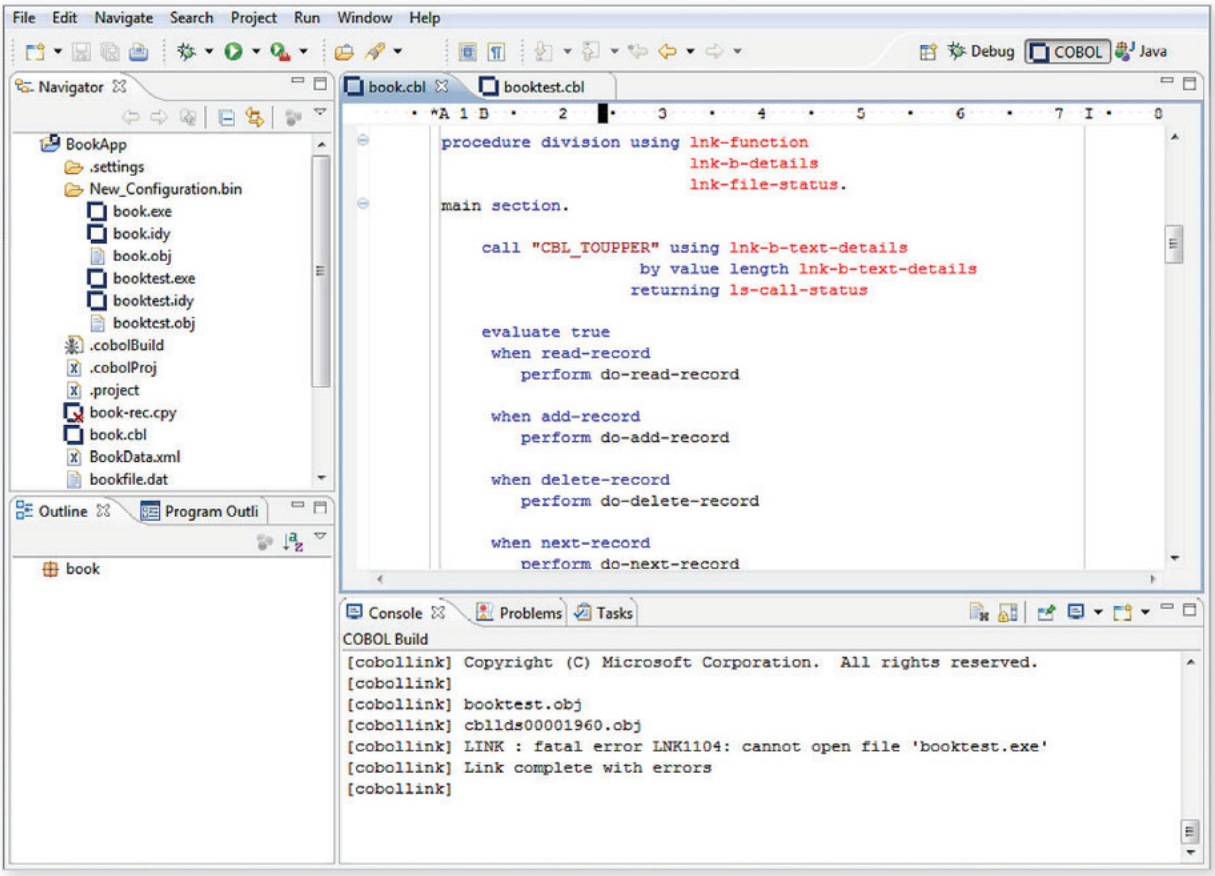
- Automatic code completion
- Syntax checking, helping you out with writing correct code while you type
- Debugging, with step-by-step, break points, variable inspection, etc.
- Navigation capability (click on an object, go to its definition; find where an object is used and the hierarchy of calls leading to it; etc.).

When using an Eclipse-based editor, the mainframe complexity is completely hidden from the developers and they can use the same Eclipse-based editor as their peer Java developers.

Another advantage is that by using one and the same Eclipse framework for both your mainframe and non-mainframe development, you can save on the costs for managing different development environments.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
EDIT       IKANALM.DEMOS.TEST.SRCBATCH(DEMO21) - 01.19          Columns 00001 00080
Command ===>                                                    Scroll ===> CSR
000012         IDENTIFICATION DIVISION.
000013         PROGRAM-ID. DEMO21.
000014        * MODEL   "DEMO21"
000015        * PROGRAM "DEMO21"
000016        *  VERSION 0001
000017        *  UPDATED BY ""IKAN\fib""
000018        *  ON 2012-03-05 08:29:34.
000019        ************************************************
000020        * GENERATED BY THE METASUITE GENERATOR
000021        *  FOR IKAN SOFTWARE
```

Mainframe: ISPF editor, displaying a COBOL program



Eclipse-based editor for a COBOL program

File System

On the mainframe

Once a COBOL, PL/1 or JCL code program is developed, it needs to be stored in a file system. On IBM z/OS, the COBOL, PL/1 and JCL code is stored in a PDS (Partitioned Data Set).

A PDS contains one or more members, whereby each member represents one file.

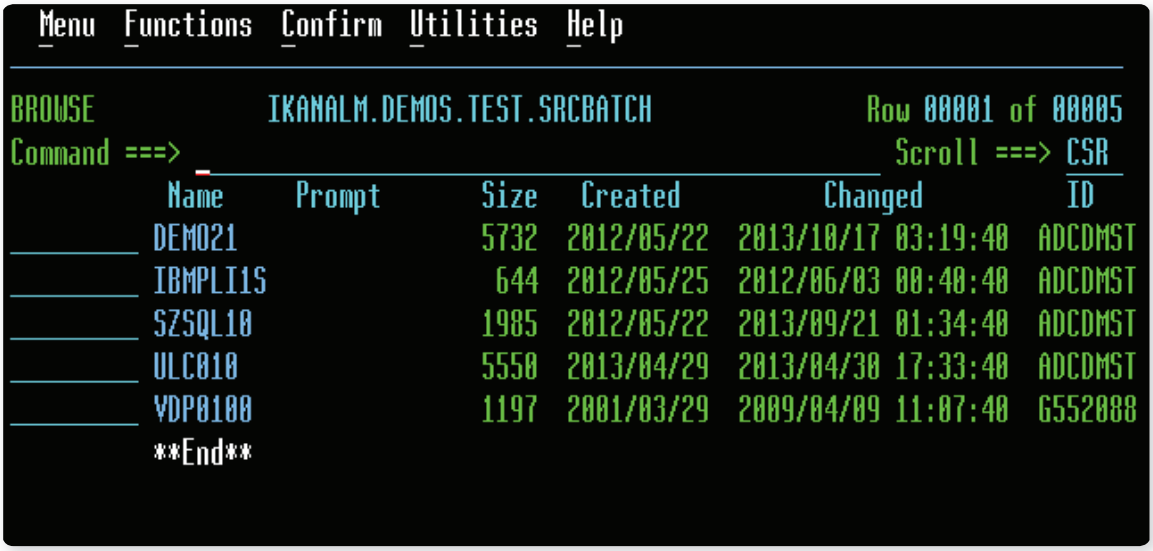
When using ISPF as editor, each individual program is saved in a PDS member.

Non-mainframe alternative

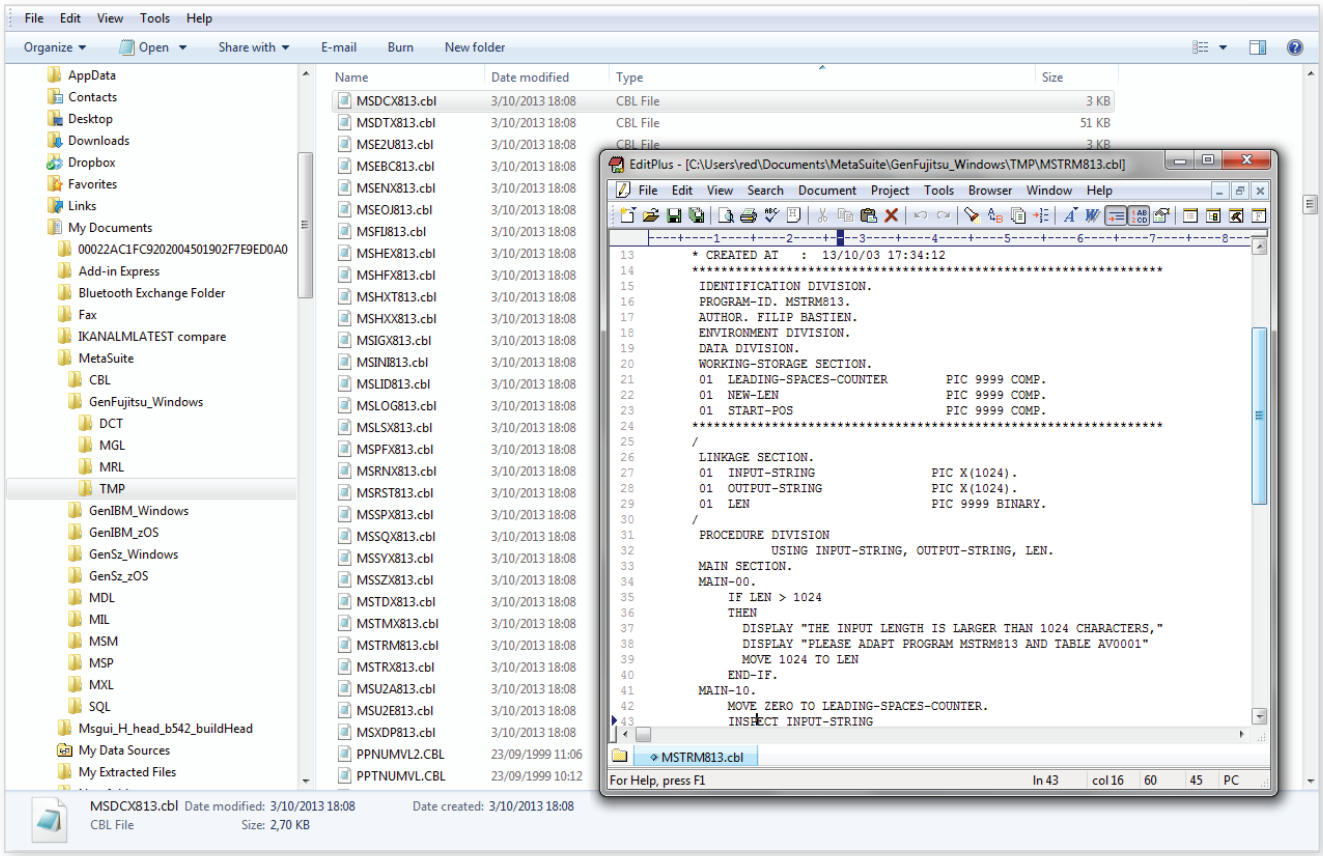
When using an Eclipse-based editor, the program is saved in a standard Windows directory or directly in a PDS through an interface with z/OS.

In Windows terminology, we talk about a file directory instead of a PDS, and members are called files. From a functional point of view there is no difference.

Using a Windows or Linux-based directory structure allows you to have one and the same file system for all of your development, be it COBOL, PL/1 or Java, which will be easier to manage and maintain.



IBM mainframe Partitioned Data Set, member list



Windows directory with one member opened

Versioning System

On the mainframe

In a PDS, you can have one copy of your program: when you change and save it with the same name, the old copy will be overwritten and you will have no history.

Although some editors can create a backup of the file before updating, this does not give the same functionality as a proper versioning system.

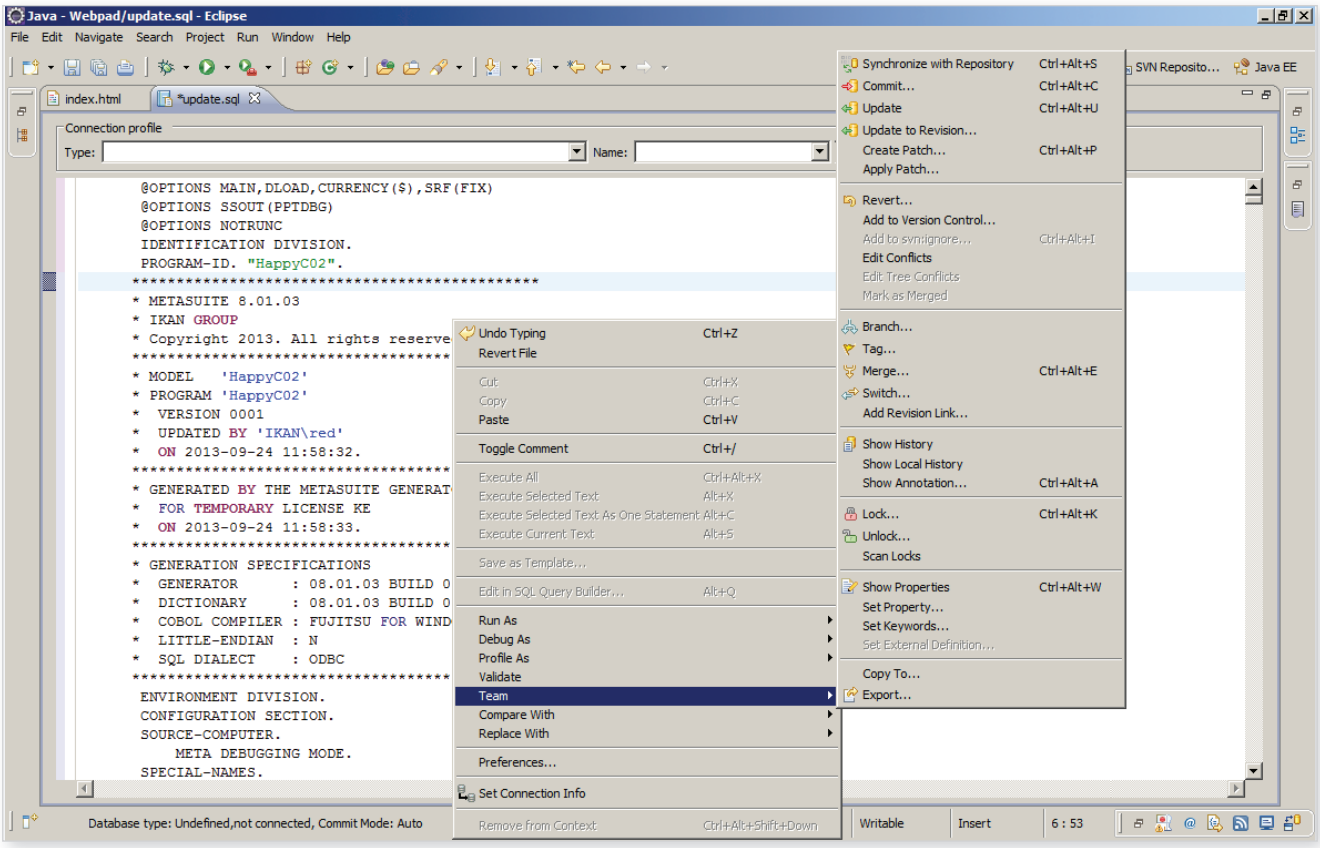
On IBM z/OS, CA Librarian, CA-Panvalet, IBM SCLM, MSP DataManager are historically the best known versioning systems. CA-Endevor and SERENA CHANGEMAN are also widely used for version control, offering additional configuration features.

Non-mainframe alternative

For non-z/OS environments, we have IBM Clearcase, CVS, Subversion, GIT and others.

All Eclipse-based editors have access to version control repositories. Eclipse has a “Team” function that allows you to connect with most of the common VCRs like CVS (standard Eclipse), Subversion (subclipse, subversive), GIT, …

Those non-z/OS based library or version control repositories provide good alternatives for the classic mainframe-based library or version control systems. As CVS, Subversion and GIT are very popular in the distributed world, using these also for mainframe versioning offers you one and the same solution for managing all your versioning, be it for mainframe, Windows, Linux or mobile.



Eclipse Team functionality

Compile Procedure

Once a program (be it in COBOL, Java or.NET) is written and added to the versioning system, it needs to be compiled or built. IBM z/OS uses the Job Control Language (JCL) as a scripting language. To write JCL, you can use ISPF, a JCL generator or, once again, an Eclipse-based editor. The example below is a standard example of a JCL to compile a COBOL program. Based on a standard script, IKAN ALM will generate the complete JCL for you and will do the necessary to submit the JCL.

```
//IGYWCLG PROC LNGPRFX='IGY.V4R1M0',SYSLBLK=3200,
//             LIBPRFX='CEE',GOPGM=GO
//*
//*  COMPILE, LINK EDIT A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IGY.V4R1M0       PREFIX FOR LANGUAGE DATA SET NAMES
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   GOPGM     GO               MEMBER NAME FOR LOAD MODULE
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL  EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//LKED   EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&&GOSET(&GOPGM),SPACE=(TRK,(10,10,1)),
//             UNIT=SYSDA,DISP=(MOD,PASS)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(10,10))
//GO     EXEC PGM=*.LKED.SYSLMOD,COND=((8,LT,COBOL),(4,LT,LKED)),
//             REGION=2048K
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*//CEEDUMP  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
```

First conclusion

For developing mainframe-based applications you can use native mainframe tools (ISPF, JCL, …), but you can just as well use the same tools as Windows, .NET and Mobile developers use:

- Eclipse-based editors for COBOL or PL/1, instead of ISPF editors,
- Standard Windows directories instead of a PDS,
- IBM ClearCase, CVS, Subversion, GIT, … as version control repositories,
- Ant or Maven as scripting languages, instead of JCL.

Main advantage: Using the same tools, considerably lowers the costs of having to manage different systems.

Once the development of your COBOL, PL/1 programs is done and your developer has committed his work to the version control repository, IKAN ALM takes care of the next steps in the application lifecycle: compile/build, deploy or promote to test or production environments.

Enterprise-wide Application Lifecycle Management

As development is just a part of the overall software release process, a complete solution, be it for mainframe of for a PC-based environment, should also cover the other parts of the release process or application lifecycle, and especially the deployment to the mainframe

Application Lifecycle Management, abbreviated as ALM, refers to the capability to integrate, coordinate and manage the different phases of the software delivery process. From development to deployment, ALM is a set of pre-defined processes that include definition, design, development, testing, deployment and management. Throughout the ALM process, each of these steps are closely monitored and controlled.

More specifically, enterprise-wide ALM needs to offer a solution for the following:

1. It should be able to manage different environments: the mainframe environment (Eclipse -based or not) as well as the non-mainframe environment, whereby also the dependencies between those different environments managed. In many cases, mainframes are used to develop and manage the back-end applications and the front-end applications are being developed and managed in Java or .NET.

2. Another important point of attention is the difference between release-based and package-based application lifecycle approaches.

a) In a PC-based environment (Java, .NET) usually all components of a project are taken into account to build or promote (deploy) a release. That is the release-based approach.

b) In a mainframe environment, not all COBOL programs that are part of a project are compiled, promoted or deployed together. Only those programs and components that have changed or that are new will be taken into account. That is the package-based approach.

Thus, to have an enterprise-wide ALM solution, you will need a product that can handle release-based (all components) and package-based (some components) "releases" and this for both mainframe and non-mainframe environments.

IKAN ALM is such a solution. It not only offers the same capabilities as mainframe-based products like CA-Endevor, Serena ChangeMan or IBM SCLM, which are pure mainframe products, but on top of that IKAN ALM is more cost-effective, straightforward to implement and manage, and supports both mainframe and non-mainframe systems.

IKAN ALM: Bringing Both Worlds Together

In the previous sections we explained how developers can use an Eclipse-based editor as an alternative for ISPF to do their work, how standard Windows directories compare to a PDS, what the different alternatives for version management are and what is generally expected from a complete Application Lifecycle Management solution.

The next question that arises is: How does IKAN ALM, being a web-based application, handle the Application Lifecycle Management process for a mainframe?

IKAN ALM Overview

The graphic on the next page shows an overview of the IKAN ALM process and its main stakeholders.

IKAN ALM offers the following key services:

- **Lifecycle per project and per branch:**
  It is up to you to define your lifecycle(s). A classic example is DTAP: Development, Test, Acceptance and Production.

- **Compile/Build:**
  IKAN ALM supports continuous integration, scheduled builds or builds on demand. For each Build, IKAN ALM can give you an overview of the related issues. Issues are entered by the developer when committing to the VCR. IKAN ALM will synchronize them with the Issue Tracking System.

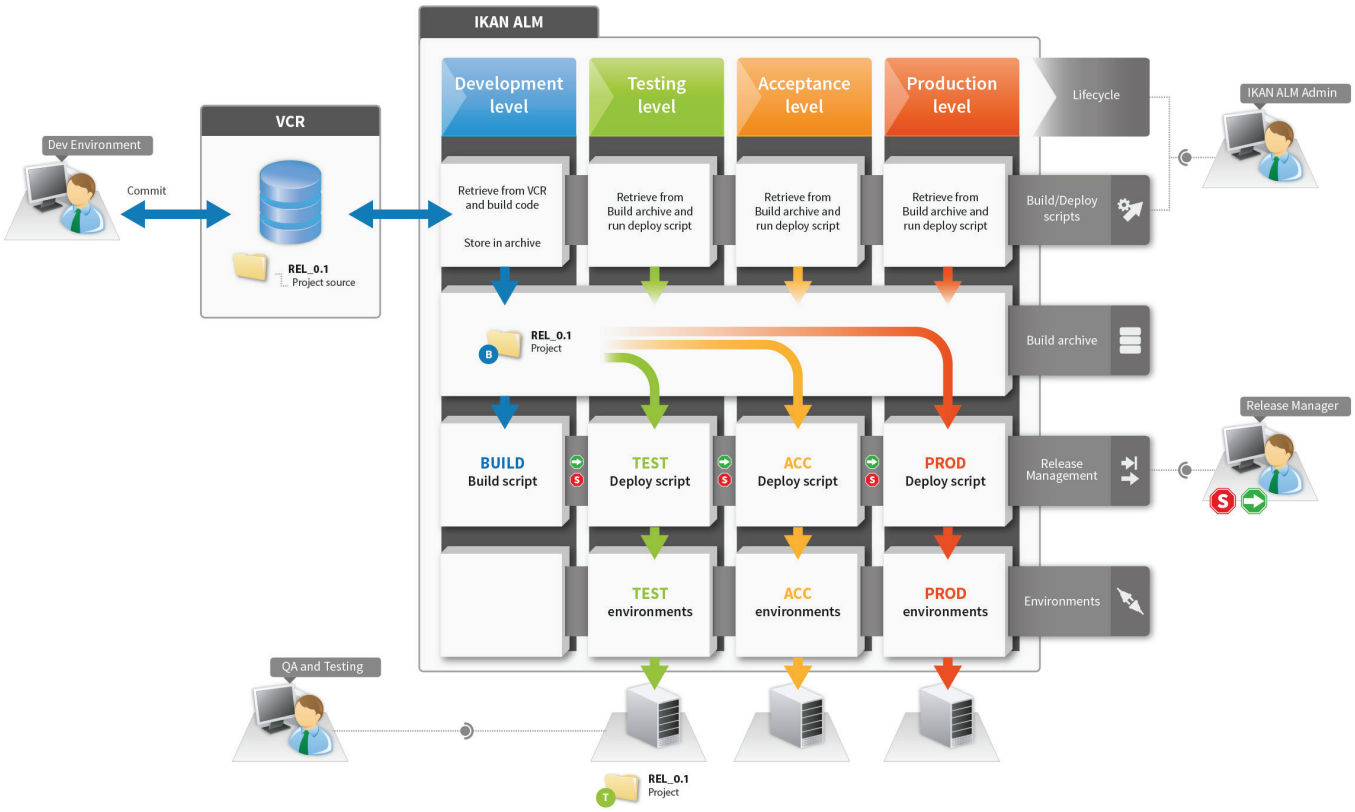
- **Deploy:**
  Once a Build is done, IKAN ALM can deploy or promote the Build result to a next level. That next level can be a test level or a production level.

- **Phases:**
  The concept of phases is one of IKAN ALM's major assets.

  Compile/Build or Deploy actions in IKAN ALM are performed by executing a sequence of Phases, each representing specific tasks or actions. The IKAN ALM core functionality is performed by so-called "Core" Phases which are read-only and form an integral part of IKAN ALM.

  Additionally, Users can create their own "Custom" Phases which can be completely adapted to their specific needs and environments. Phases can be reused and shared between different Projects and even different IKAN ALM installations.

- **Approval and Notification:**
  Any deploy request within IKAN ALM can depend on an approval. An email will be sent to the approver, and the approver can then approve or disapprove. Any request can be the subject of a notification: through notifications you will be informed if an action was successful or unsuccessful.



Next to these key services, IKAN ALM offers:

- **Integration with Version Control Systems:**
  For each project you can specify the VCR to be used by IKAN ALM. Different VCRs can be used for different projects.

- **Integration with Issue Tracking Systems:**
  IKAN ALM comes with an Issue Tracking System plugin that you can customize to your own needs.

  Per Build or Deploy, the related issues are displayed, and with a simple click, the issue Tracking System will be opened to show all the related information. This integration is bi-directional.

IKAN ALM Architecture

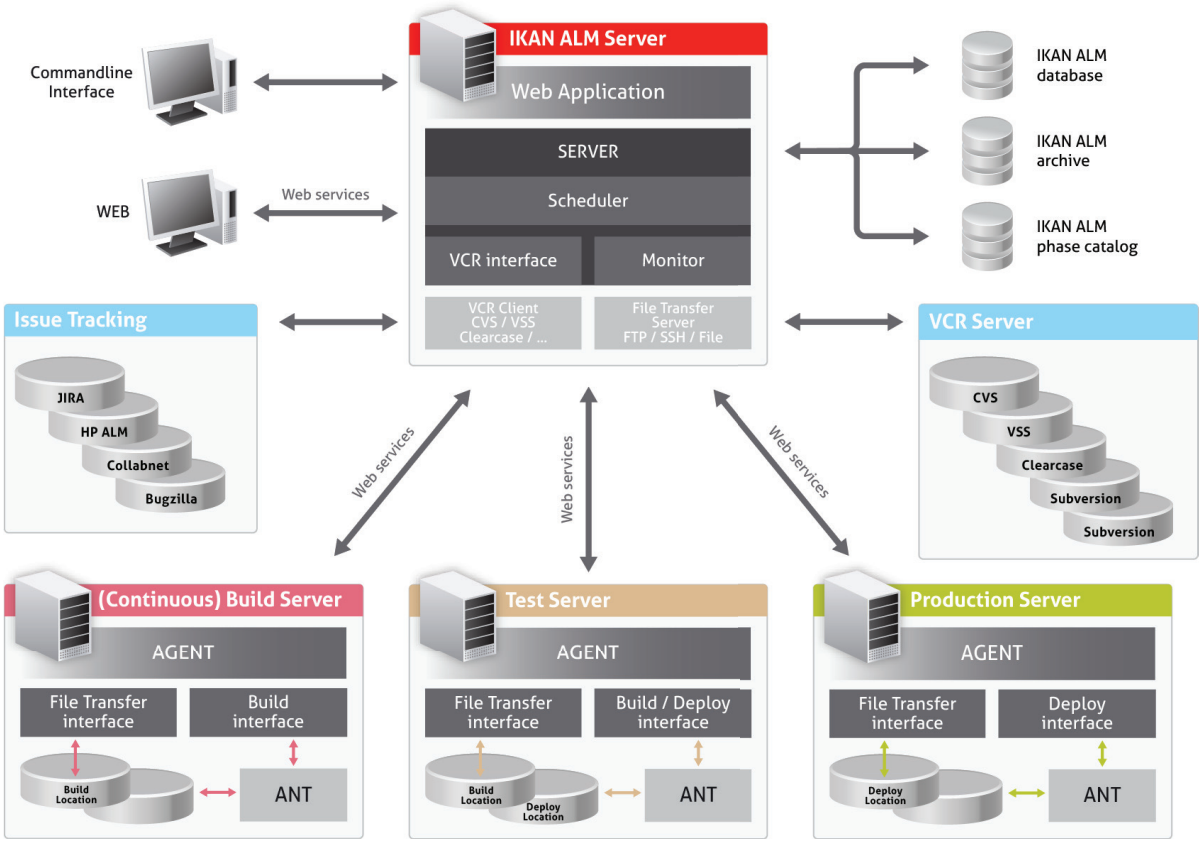
The graphic on the next page shows the IKAN ALM architecture.

- **Web-based:**
  IKAN ALM is a web-based application. All you need is a web browser.

- **Agents:**
  IKAN ALM works with Agents: Compile/Build or Deploy requests can be executed on a local or on a remote agent.

- **Communication:**
  IKAN ALM works with ftp, file copy or secured shell.

- **Reporting:**
  IKAN ALM has an Open Source based reporting module which is easily extendible.



Conclusion

At first sight, mainframe development seems to be completely different from .NET or Java development. In both cases, however, you need the same building blocks: a program editor, a file system, a version control repository and a compile(build) procedure.

As technology constantly evolves and applications are more and more used on different platforms, it becomes inevitable to combine the best of both the mainframe and the non-mainframe world.

IKAN ALM offers an alternative for pure mainframe-based development by combining an Eclipse-based development environment with a distributed Version Control Repository. On top of that IKAN ALM complements the development process with Application Lifecycle Management and Deploy services.

IKAN ALM's major asset is its concept of Phases. JCL can be very complicated. By using the IKAN ALM Phases, you can easily generate and tailor any JCL step.

Thanks to the phase concept and the available models and resources, we can also guarantee an easy and successful implementation (as an average, it will only take a few weeks). The key requirement is for you to define your ALM process. Once that has been established, the implementation of IKAN ALM is fast and straightforward.

If you are already using a mainframe solution like CA-Endevor or Serena ChangeMan, and you would decide to migrate to IKAN ALM, you will of course also need to migrate your CA-Endevor or Serena ChangeMan legacy to IKAN ALM. To do so, we have also worked out detailed migration procedures.

For more technical details on how IKAN ALM works and what the different tasks are for Users, Global Administrators and Project Managers, we refer to our technical white paper “Integrating IKAN ALM and Mainframes”. That document is intended for your developers, technical mainframe and/or non-mainframe experts and software architects. We are confident that, after having read that document, they will confirm you the enormous advantages of putting in place our IKAN ALM solution.

In a nutshell: by implementing IKAN ALM, you can continue exploiting the full strengths of your mainframe and seamlessly combine them with new innovative tooling.

This will help you cutting down the costs of maintaining different systems, and above all ease the work of your developers as IKAN ALM will take care of the different steps in your application’s lifecycle including its deploy on the mainframe.

For More Information

To know more, visit http://www.ikanalm.com
Contact IKAN Development: info@ikanalm.com

Related Document

Integrating IKAN ALM and Mainframes

IKAN Development N.V.
Kardinaal Mercierplein 2
2800 Mechelen
Tel. +32 15 797306
info@ikan.be
www.ikan.be

© Copyright 2013 IKAN Development N.V.

The IKAN Development and IKAN ALM logos and names and all other IKAN product or service names are trademarks of IKAN Development N.V. All other trademarks are property of their respective owners. No part of this document may be reproduced or transmitted in any form or by any means, electronically or mechanically, for any purpose, without the express written permission of IKAN Development N.V.

IKAN